CALIFORNIA 93943

T218325

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

A MICROCOMPUTER BASED
GENERAL LINEAR PROGRAMMING
OPTIMIZATION PACKAGE

by

Donald W. Theune

September 1983

Thesis Advisor: G. G. Brown

Approved for public release; distribution unlimited.

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER

2. GOVT ACCESSION NO.

3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle)

A Microcomputer Based General Linear Programming Optimization Package

5. TYPE OF REPORT & PERIOD COVERED
Master's Thesis; September 1983

6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s)

Donald W. Theune

8. CONTRACT OR GRANT NUMBER(s)

9. PERFORMING ORGANIZATION NAME AND ADDRESS

Naval Postgraduate School
Monterey, California 93943

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS

11. CONTROLLING OFFICE NAME AND ADDRESS

Naval Postgraduate School
Monterey, California 93943

12. REPORT DATE
September 1983

13. NUMBER OF PAGES
73

14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)

15. SECURITY CLASS. (of this report)
Unclassified

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

microcomputer
linnear programming
optimization

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The importance of mathematical models as tools in decision making has motivated increased interest in that theory and its implementation. This paper describes fundamental techniques of linear programming which have been combined to offer a microcomputer based optimization package. The package is machine portable and will accept input from files created by other programs. Thus the

20. ABSTRACT (Continued)

package affords the opportunity to build a mathematical programming system based on its ability to solve bounded variable linear sub-problems. Written in JRT PASCAL 3.0 and implemented on a portable, 8-bit microcomputer (KAYPRO-II), this package places the fundamental tool of optimization in the office, classroom and home.

Approved for public release; distribution unlimited.

A Microcomputer Based General Linear Programming Optimization Package

by

Donald W. Theune
Major, United States Marine Corps
B.A., Lakeland College, Sheboygan, Wi., 1978

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN OPERATIONS RESEARCH

from the

NAVAL POSTGRADUATE SCHOOL

September 1983

ABSTRACT

The importance of mathematical models as tools in decision making has motivated increased interest in that theory and its implementation. This paper describes fundamental techniques of linear programming which have been combined to offer a microcomputer based optimization package. The package is machine portable and will accept input from files created by other programs. Thus the package affords the opportunity to build a mathematical programming system based on its ability to solve bounded variable linear sub-problems. Written in JRT PASCAL 3.0 and implemented on a portable, 8-bit microcomputer (KAYPRO-II), this package places the fundamental tool of optimization in the office, classroom and home.

TABLE OF CONTENTS

I. INTRODUCTION ---------- 9

II. PROBLEM REPRESENTATION ---------- 14

A. CANONICAL FORM ---------- 14

B. BASIS DEFINITION ---------- 15

C. MAXIMUM VS. MINIMUM ---------- 16

D. SLACK LOGICAL VARIABLES ---------- 16

E. SURPLUS LOGICAL VARIABLES ---------- 17

F. ARTIFICIAL VARIABLES ---------- 18

G. VARIABLE BOUNDS ---------- 18

H. VARIABLES WITH UNRESTRICTED SIGN ---------- 19

III. ALGORITHM SELECTION ---------- 21

A. A GENERAL SIMPLEX TECHNIQUE ---------- 21

1. Rewrite (LPC) in Terms of Basic Solution ---------- 22

2. Basic Solution ---------- 22

3. Priceout ---------- 23

4. Ratio Test ---------- 24

5. Reflection ---------- 25

6. Update ---------- 26

B. TWO PHASE SIMPLEX ---------- 26

C. A BOUNDED VARIABLE SIMPLEX TECHNIQUE -------- 28

1. Transformation of Variables ---------- 28

2. The Algorithm ---------- 30

D. REVISED SIMPLEX--PRODUCT FORM OF THE INVERSE ---------- 32

1. Advantages ---------- 32

2. Elementary Matrices ---------- 33

E. DEGENERACY AND CYCLING ---------- 35

F. REINVERSION ---------- 39

IV. IMPLEMENTATION ---------- 44

A. DATA STRUCTURES ---------- 45

B. INPUT ---------- 47

1. TYPEPROB ---------- 48

2. MODPROB ---------- 51

3. READPROB ---------- 52

C. THE PROBLEM SOLVER ---------- 53

1. SIMPLEX ---------- 53

2. PHASEI ---------- 53

3. BTRAN ---------- 54

4. CHUZQ ---------- 54

5. FTRAN ---------- 54

6. CHUZP ---------- 54

7. PIVOT ---------- 55

8. REINVERT ---------- 55

D. OUTPUT ---------- 55

V. CONCLUSIONS AND RECOMMENDATIONS ---------- 57

APPENDIX A: EXAMPLE INPUT FILE ---------- 60

APPENDIX B: INTERACTIVE SESSION ---------- 62

APPENDIX C: EXAMPLE PROBLEM FILE ---------- 68

APPENDIX D: BTRAN PROCEDURE LISTING ---------- 69

APPENDIX E: OUTPUT LISTING FOR LP PROBLEM SOLVER ----- 70

LIST OF REFERENCES ---------- 72

INITIAL DISTRIBUTION LIST ---------- 73

ACKNOWLEDGEMENT

I am grateful to Professor Larry Bodin of the University of Maryland for his patience and help in the earliest stages of this effort. Problem formulations provided by Professor Bodin aided immeasurably in decreasing debugging time associated with the implementation of algorithms. My thanks, also, to Major Lee Dewald, USA, and Lieutenant J.S. "TEX" Moore, USN, whose insights and interest in the topic led to a more precise understanding of the theory. I would like to express a sincere thank you to Professor Jerry Brown without whose guidance and direction this project would never have been completed. Finally, I wish to thank my wife Julia for her patience and understanding, and my children Jason and Jennifer for their cooperation.

I. INTRODUCTION

In the past 35 years, thousands of papers have been published dealing with decision theory and optimization. In the development of that theory, the study of solutions to systems of linear inequalities has played a large role. The importance of mathematical models as tools in decision-making has motivated increased interest in that theory and its implementation.

Linear programming is the minimization of a linear function subject to linear inequality constraints [Ref. 1]. The techniques applied to this area of optimization are rooted in the theory of solutions to systems of linear inequalities and the mathematics of linear equation theory. Applications of this theory provide insight into the problem of minimizing a convex function whose variables must satisfy a system of convex inequality constraints. The applications also provide a framework for extending problems in mathematical statistics and a foundation upon which are built modern algorithms for the solution of optimization problems whose variables are integer valued (integer programming) or whose constraints are non-linear (non-linear programming). Due to this wide range of applications, the availability of efficient implementations of linear programming algorithms has become important. Routines to solve a wide variety of

problems have been developed for use on mainframe computers. Many competitive businesses and industries routinely rely on such programs to assist in day-to-day corporate decision making. These systems tend to be expensive to operate due to overhead costs associated with the operation of large-scale computer hardware and the training of operators of time-share systems.

The microcomputer is fast becoming a viable alternative to time-share mainframes as a source of computational power for small businesses and individuals. Rapid advances in computer technology, especially in micro-electronics, have made possible the routine use of many of the basic, theoretical algorithms which were previously viewed as too complex and inefficient. The incorporation of these advances into the manufacture of "micros" and the development of microcomputer-based programming languages responsive to user needs have only recently allowed for implementation of fundamental optimization tools on the relatively inexpensive microcomputer. Obvious trade-offs arise in the comparison of "micros" to mainframes. While the speed at which large computers accomplish computational results is surely their greatest asset, so are size and the costs of hardware and software their greatest liabilities. On the other hand, the relatively inexpensive one-time purchase price of the microcomputer and its associated software must be somewhat offset by its slower computational speed and smaller memory size.

Another disadvantage of the microcomputer is that efficient, "user-friendly" optimization systems are not currently available for the microcomputer environment. This is possibly due, in part, to the intense effort required to develop sophisticated microcomputer software in this severely restricted environment. While a few basic implementations of linear programming optimization theory are beginning to appear, no reference to any work of consequence has been found in the literature. Further, it is not clear that the advantages of the microcomputer have been fully exploited in those few systems currently being released. In fact it is apparent that so-called microcomputer based systems are usually weaker versions of systems designed and implemented for mainframes from simple textbook descriptions and modified for use on a microcomputer; the elementary theory incorporated limits problem size and solution efficiency due to insensitivity to the strengths and weaknesses of the target computer. Few of the sophisticated features of widely available commercial, mainframe optimization systems have been transferred to "micros".

The following is an attempt to amalgamate current technology and fundamental theory regarding linear programming and to create an easy-to-use, interactive computer program with wide applicability on current state-of-the-art microcomputers. The system described is designed for a most restrictive "eight-bit" microcomputer in the hope that

upward compatability with larger machines and new-generation computers will allow more advanced capabilities as technology admits in the future. The algorithms described and implemented range in complexity from simple algebraic manipulation, as in the simplex pivot, to the more involved technique of the preassigned pivot procedure initially developed by Hellerman and Rarick [Ref. 2]. Subtle modifications have been made to many of the basic algorithms to allow implementation on the microcomputer. Other algorithms have been embedded in the code and are activated where required, e.g., a new modification of Bland's first rule for the avoidance of cycling in the presence of degeneracy [Ref. 3]. In the section dealing with the use of the program, the input and output routines are discussed.

The package employs a reasonable number of "large-scale" optimization features, including sparce problem representation and product form inverse. The limits on the size of problems that these programs can cope with is principally dependent on the size and bit density of the off-line storage available and the size of internal random access memory (RAM). Numerous vectors are implemented whose dimensions are problem dependent and allocated at run-time, providing true dynamic dimensioning. "In core, out of core" operation maximizes RAM utilization in a fashion reminiscent of second generation mainframe computers two decades old.

JRT Pascal version 3.0 is used to implement the theory discussed in this thesis. The program modules are compiled

on the KAYPRO II, eight-bit microcomputer. The associated algorithm is designed based on speed (number of calculations required), overhead required, available programming language constructs, and simplicity of theory; usually in that order.

Wide-spectrum stress testing of the package has not been performed due to time constraints.

II. PROBLEM REPRESENTATION

In this chapter we define a canonical problem format (LPC) and the terms to be used throughout the development of the algorithm. We then show that, given any problem formulation, (LPF), an equivalent, canonical form may be achieved through the application of simple transformations and/or the introduction of additional logical variables such that a solution to (LPF) can always be constructed from an equivalent solution to (LPC).

Any linear program is formulated as follows:

(LPF) minimize (or maximize) cx
subject to linear equality or inequality constraints with variables,

$x_j \geq 0$ non-negative,

$x_j \leq 0$ non-positive, or

x_j unrestricted in sign ("free").

A. CANONICAL FORM

The algorithm developed in this thesis is designed to provide optimal solutions to linear programming problems stated in the following "canonical" form (assuming an optimal solution exists):

(LPC) minimize cx ("objective function")

subject to

$$Ax = b, \qquad \text{("constraints")}$$

$$0 \leq x \leq UB, \qquad \text{(UB are "upper bounds")}$$

$$b \geq 0; \qquad \text{("non-negative right hand side")}$$

where x is an n-dimensional column vector, c is an n-dimensional row vector, A is an $m \times n$ matrix of technological coefficients with $n \geq m$, and b is an m-dimensional column vector. All (LPF) forms may be converted to the (LPC) form as described in the following sections.

B. BASIS DEFINITION

A basis, denoted by B, is any set of m linearly independent columns of A. An initial basis, B_0, is the first basis considered in the iterative solution of (LPC). It will be convenient to construct an initial basis, during the process of transforming (LPF) to (LPC), such that this initial basis consists of unit vectors, i.e., $B_0 = I$. In so doing, the inverse of the initial basis, B_0^{-1}, is identically I and its "product-form", the implicit representation of B_0^{-1} as the product of elementary transformation matrices, is trivial.

C. MAXIMUM VS. MINIMUM

Suppose that the problem (LPF) is stated as:

maximize cx

s.t.

canonical form (LPC) constraints.

Since (LPC) requires formulation as a minimization it will be necessary to transform the objective function, maximize cx. The resulting, equivalent minimization function is:

minimize (-c)x; we redefine c accordingly in (LPC).

Similarly, for each $b_i < 0$ we can perform a transformation to replace with $b_i > 0$ as follows:

i. multiply row i by -1.

ii. replace the right hand side element so that $b_i \leftarrow -b_i$.

D. SLACK LOGICAL VARIABLES

We must be concerned with transforming inequality constraints to the canonical form. Consider the (LPF) problem with the "ith" constraint of the following form:

$$a_{i1}x_1 + a_{i2}x_2 + \ldots + a_{in}x_n \leq b_i .$$

To convert this problem to canonical (LPC) form we rewrite the constraint as follows:

$$a_{i1}x_1 + a_{i2}x_2 + \ldots + a_{in}x_n + s_i = b_i, \quad s_i \geq 0,$$

where the non-negative variable, s_i, introduced is referred to as a "slack variable", or "logical plus type". Note that if some x satisfies the (LPF) constraint, it also satisfies the (LPC) constraint and vice versa. The slack variable simply takes on the value required to maintain the equality in (LPC).

E. SURPLUS LOGICAL VARIABLES

Suppose that the ith constraint from (LPC) is of the form:

$$a_{i1}x_1 + a_{i2}x_2 + \ldots + a_{in}x_n \geq b_i, \quad b_i > 0,$$

then

$$a_{i1}x_1 + a_{i2}x_2 + \ldots + a_{in}x_n - w_i = b_i, \quad w_i \geq 0.$$

The w_i variable appended here is called a "surplus variable" or "logical minus type" and again, any x satisfies the (LPF) form if and only if it satisfies the (LPC) form. Here w_i takes on the value required to maintain the equality.

F. ARTIFICIAL VARIABLES

Considering the introduction of the variable, w, into (LPC) for the previous case, we find that its coefficients form the negative of the i-th unit vector. We would prefer to have available an initial basis composed exclusively of positive unit vectors. By construction we will add a second, non-negative variable to the previous equation as follows:

$$a_{i1}x_1 + a_{i2}x_2 + \ldots + a_{in}x_n - w_i + z_i = b_i, \quad z_i \geq 0.$$

The variable introduced is called an artificial variable. Note that $z_i > 0$ in (LPC) implies that the i-th constraint in (LPF) is *not* satisfied. However, if a solution to the new problem can be found such that $z_i = 0$, then the solution to the new problem will be consistent with the solution to (LPF).

By similar convention we introduce an artificial variable into equality constraints of (LPF) to produce an initial identity basis for (LPC). Note that the values of artificial variables in (LPC) gauge the magnitudes of respective constraint violations in (LPF).

G. VARIABLE BOUNDS

Variable bounds are elementary constraints of the form

$$LB_j \leq x_j \leq UB_j .$$

Frequently, such constraints are present in (LPF) and can be expressed as variable bounds in (LPC). Variable bounds are accommodated with great efficiency in the bounded variable simplex method, but they must be identified explicitly prior to the solution.

As a further simplification, the lower variable bounds may be changed to be zero by the simple transformation (change of origin):

$$(LPC) \leftarrow (LPF)$$

$$x_j \leftarrow x_j - LB_j .$$

The resulting variables in (LPC) have

$$0 \leq x_j \leq MUB_j = UB_j - LB_j .$$

Recovery of the final solution to (LPF) from a solution to (LPC) is immediate by reversing the transformation:

$$(LPF) \leftarrow (LPC)$$

$$x_j \leftarrow x_j + LB_j$$

H. VARIABLES WITH UNRESTRICTED SIGN

An easy technique available to deal with "free", or "unrestricted variables" replaces the free variable with

the difference of two non-negative surrogates, $x_j = u_j - v_j$. Although this technique introduces an additional variable, the overhead cost is only the duplication of the coefficients of x to create the additional variable, and a final transformation to recover the resulting value of x_j, which is relatively inexpensive. Storage overhead for the new variable is of little consequence due to the use of the "out-of-core" storage of non-zero problem elements.

All complications in (LPF) may now be handled by standard elementary problem transformations to produce (LPC). Thus x may be unrestricted in sign, and arbitrary inequality constraints can be accommodated so that any formulation may be easily modified to produce (LPC).

III. ALGORITHM SELECTION

Once the problem formulation has been transformed to canonical form (LPC), an algorithm must be developed, or adapted, that provides the technique required to solve the (LPC) problem. This algorithm must be both efficient, and sympathetic to the eccentricities of the small computer. In keeping with common practice on microcomputers, we adopt a straightforward, elementary textbook approach to the mathematical justification of our simplex algorithm. The implementation is necessarily more sophisticated. The general algorithm that will be discussed here is the two-phase, revised simplex method (simplex method using multipliers) [Ref. 1]. Specifically, the non-zero elements of the product form of the inverse and all non-zero problem elements will be stored in two random access diskette files and will be read into internal memory only when required. The associated reinversion technique used is the preassigned pivot procedure [Ref. 2]. Each section of the algorithm will be discussed in this chapter. The next chapter will be devoted to the associated implementation.

A. A GENERAL SIMPLEX TECHNIQUE

Assuming that the problem statement has been transformed to the form of (LPC), we may proceed as follows:

1. Rewrite (LPC) in Terms of Basic Solution

At any iteration; let B be a set of m linearly independent columns of A which leads to the column partition of (LPC):

$$\min \quad cx$$

$$\text{subject to} \quad Ax = b \Rightarrow [B|N]\, x$$

$$= BX_B + NX_N = b .$$

The variables X_B are called basic (or dependent) with coefficients B, and the variables X_N are referred to as non-basic (or independent) variables with coefficients N. It will be convenient to index columns of B and the basic variables X_B by the row to column mapping j_i (for row i, the associated basic column and variable is j_i).

2. Basic Solution

A basic solution at any iteration consists of the values of X_B and X_N. The value of each of the X_N variables is, by convention, a constant equal to zero or the associated upper bound. For illustration, we will assume all non-basic variables are at zero.

The values of the basic variables X_B are defined by the problem statement (LPC) as follows:

$$\text{Since} \quad BX_B + NX_N = b$$

$$\text{then} \quad X_B = B^{-1}b - B^{-1}NX_N$$

$$\text{and} \quad X_N \equiv 0 \Rightarrow X_B = B^{-1}b .$$

The value of the objective function is defined:

$$cx = c_B X_B + c_N X_N$$

$$= c_B B^{-1} b + c_N X_N$$

and again

$$X_N \equiv 0 \Rightarrow cx = c_B B^{-1} b .$$

3. Priceout

The purpose of the "priceout step" is to identify a non-basic variable for which the rate of improvement in the value of the objective function is favorable.

Consider:

$$cx = c_B X_B + c_N X_N$$

$$= c_B (B^{-1} b - B^{-1} N X_N) + c_N X_N$$

$$= c_B B^{-1} b + (c_N - c_B B^{-1} N) X_N$$

$$= c_B B^{-1} b + r X_N .$$

Note that $c_B B^{-1} b$ is the current value of the objective function given $X_N \equiv 0$ and $(c_N - c_B B^{-1} N)$ is the vector of reduced costs, r, which indicates how much the objective function changes as X_N changes, This vector, r, will be

searched to determine which variable, x_q, will enter the basis. The subscript, q, indexes the column chosen.

If no favorable price is found in this search, an optimal solution is declared for the current objective.

4. Ratio Test

The "ratio test" insures that the subsequent solution will continue to satisfy all variable bounds. The ratio test searches for the first variable to reach one of its bounds as x_q increases in magnitude. If basic variable x_{j_i} is the variable which first reaches a bound, 0 or UB_{j_i}, in terms of the incoming variable x_q, then x_{j_i} leaves the basis and x_q enters. Otherwise, x_q reaches its own opposite bound before any basic variable does, and x_q will not enter the basis.

Let $Y = \{y_{ij}\}$ represent the updated values of the elements of the matrix A, with Y_j denoting column j, such that:

$$Y_j = B^{-1}A_j \quad \text{"updated column"},$$

$$Y_0 = B^{-1}b \quad \text{"right hand side"} .$$

and recall that the basic variables are determined by:

$$\begin{aligned} X_B &= B^{-1}b - B^{-1}NX_N \\ &= B^{-1}b - B^{-1}A_q x_q \\ &= Y_0 - Y_q x_q = X_B(x_q) . \end{aligned}$$

Then the variable which exits the basis is determined by:

$$\text{MIN} \quad \begin{cases} \text{a)} & UB_q, \text{ upper bound on entering variable} \\ \text{b)} & \min y_{i0}/y_{iq}, \quad y_{iq} > 0 \\ \text{c)} & \min(y_{i0} - UB_{j_i})/\, y_{iq}, \quad UB_{j_i} \text{ finite and } y_{iq} < 0. \end{cases}$$

Let p be the index of the constraint in which the "winning" ratio is found so that $p = \Phi$ in case a), and x_{j_p} is the leaving variable in cases b) or c). If $p = \Phi$ and $UB_q = \infty$, then the problem is unbounded: no variable reaches a finite bound as x_q increases and the value of the objective function increases without limit.

5. Reflection

We have assumed for convenience that $X_N \equiv 0$. Suppose that x_j is bounded so that $0 \leq x_j \leq UB_j$, and that at some point x_j leaves the basis at its upper bound, UB_j. Then to preserve $X_N \equiv 0$ and avoid complication in the logic required to handle this new type of variable, we "reflect" the variable x_j.

The variable x_j is replaced by $UB_j - x_j$. We will record this replacement as a status of the variable x_j (noting that another reflection of x_j restores its initial status), and take care to modify the right hand side with the constants $A_j UB_j$ and treat column j as if its sign were reversed.

6. Update

For cases a) and c) of the ratio test, a reflection is required in the update. For case a), no further work need be done as x_q remains non-basic and B is unchanged. Otherwise, in cases b) and c), the variable x_q enters the basis and x_{j_p} leaves the basis. In these cases, an elementary transformation matrix E_k must be formed such that $E_k B_k^{-1} = B_{k+1}^{-1}$, where B_{k+1}^{-1} is the inverse of the new basis with $x_q \in x_B$ and $x_{j_p} \in X_N$. In this way a "pivot" is performed about the element y_{pq}, with:

$$E_k = \begin{bmatrix} 1 & 0 & \cdots & 0 & v_1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 & v_2 & 0 & \cdots & 0 \\ & 0 & & & & \cdot & & \cdot \\ & \cdot & & 1 & & \cdot & & \cdot \\ & \cdot & & & v_p & \cdot & & \cdot \\ & \cdot & & & & 1 & & 0 \\ 0 & 0 & & 0 & v_m & 0 & \cdots & 1 \end{bmatrix}$$

where $v_i = -y_{iq}/y_{pq}$, $i \neq p$ and $v_p = 1/y_{pq}$.

B. TWO PHASE SIMPLEX

If the initial basis includes artificial variables, the initial basic solution may be infeasible. The values of these variables must be reduced to zero to obtain a feasible basic solution, if one exists. Phase I of the two-phase simplex method accomplishes this task. If Phase I produces a feasible solution, the original (LPC)

objective function is restored and Phase II is begun. Phase II improves a basic feasible solution to optimality.

The first step in Phase I is to introduce a special objective function. This vector contains a zero for each non-artificial variable and positive unity as a penalty for each artificial variable in the initial basis, so that the objective function is of the form:

Phase I objective:

MIN (sum of the artificial variables).

The simplex technique is then applied until the priceout step produces no favorable (incoming) variable. At this point an optimal solution to the Phase I problem is obtained. If the optimal value of the Phase I objective function is zero, the artificial variables all have zero value and the associated basis provides a feasible solution to the original problem (although not necessarily optimal for (LPC)). If, on the other hand, the final value of the objective function is positive then not all of the artificial variables have a value of zero. Thus a feasible solution to (LPC) does not exist, and hence none exists for (LPF).

During Phase I, an artificial variable, once removed from the basis, is never allowed to re-enter. When (LPC) contains redundant constraints, artificial variables may, with value zero, remain as part of the basic set of variables

at the end of Phase I [Ref. 1, p. 103]. It is necessary that their values in Phase II never exceed zero. This is accomplished by eliminating all non-basic variables whose reduced costs, r_j, at the end of Phase I are greater than zero, for if one of these variables were introduced into the basis during Phase II, the value of some basic artificial variable would increase and the solution would again become infeasible. Once this task is completed we are guaranteed that if artificial variables form part of the basic set of variables in the various iterations of Phase II, their values will never exceed zero. (See proof; [Ref. 1, p. 103].) Also, during Phase II, an artificial variable is never allowed to re-enter the basis.

C. A BOUNDED VARIABLE SIMPLEX TECHNIQUE

1. Transformation of Variables

A "bounded-variable" problem may include non-zero lower bounds on values of variables. Since the algorithm adopted assumes all lower bounds are zero, a transformation of variables must be carried out prior to the simplex routine:

$$\hat{x}_j = x_j - LB_j$$

and requires the following additional bookkeeping:

a. For all variables with non-zero lower bounds, LB, modify the upper bound, UB, such that the modified upper bound MUB = UB - LB.

b. Modify all right hand side values such that

$$b'_i = b_i - \sum_j a_{ij} * LB_j \text{ , for all } i.$$

A retransformation will be required to express the solution in terms of the original problem statement. Let b' represent the right hand side value of the transformed problem at completion of the simplex algorithm, and $\hat{x}_j$ represent the value of the j-th transformed variable. Then the transformations required to arrive at the solution to the original problem statement are as follows:

If $\hat{x}_j$ is non-basic then $x_j = LB_j$ or $(MUB_j + LB_j)$ dependent on whether the variable x_j is at its lower or upper bound.

If $\hat{x}_j$ is basic then the following cases apply:

CASE	(LPF) bounds	(LPC) transformation
1	$0 \leq x_j < \infty$	$x_j = \hat{x}_j$
	All logical variables are bounded this way.	
2	$-\infty < x_j \leq 0$	$x_j = -\hat{x}_j$
3	x_j is a free variable	$x_j = u_j - v_j$

4	$a \leq x_j \leq b$	
	and $\hat{x}_j$ is not reflected	$x_j = \hat{x}_j + LB_j$
	or $\hat{x}_j$ is reflected	$x_j = MUB_j + LB_j - \hat{x}_j$
5	$a \leq x_j < \infty$	$x_j = \hat{x}_j + LB_j$
6	$-\infty < x_j \leq a$	$x_j = -(\hat{x}_j + LB_j)$

2. The Algorithm

The bounded variable simplex algorithm implemented is as follows [Ref. 4, p. 51].

STEP 1: (PRICEOUT) Determine the non-basic variable, x_q, for which

$$\text{MIN } (r_q = c_q - c_B B^{-1} N_q,\ r_q < 0) .$$

If no such variable exists, stop; the current solution is *optimal*.

STEP 2: (RATIO TEST) Evaluate the three numbers associated with variable x_q chosen in Step 1.

a. UB_q (this bound may be infinite)

b. $\min y_{i0}/y_{iq},\ y_{iq} > 0$

c. $\min(y_{i0} - UB_{j_i})/y_{iq},\ y_{iq} < 0,\ UB_{j_i}$ finite

where UB_{j_i} is the upper bound associated with the variable that is basic for constraint i. Note that if the upper bounds are infinite and there are no tests of type b then the ratio test may fail. In this case the solution to the problem is *unbounded* in terms of the incoming variable.

STEP 3: (UPDATE) Depending on which item in Step 2 is smallest, update as follows:

a. Reflect x_q. The variable x_q goes to its opposite bound. Subtract UB_q times column q from the right hand side. Multiply column q by -1 and change the sign of the indicator vector element e_q to show that x_q has been reflected (changed to its opposite bound). No pivot is required.

b. Let p be the minimizing index in (b) of Step 2. Then the p-th basic variable returns to its old bound. Pivot on the element in row p and column q.

c. Let p be the minimizing index in (c) of Step 2. Then the p-th basic variable goes to its opposite bound. Reflect x_{j_p}. Subtract UB_{j_p} from y_{p0}, where UB_{j_p} is the upper bound associated with the variable that is basic for row p; reverse the signs of y_{pj_p} and e_{j_p} (to show the reflection) and pivot on the element in row p and column q.

RETURN TO STEP 1.

D. REVISED SIMPLEX--PRODUCT FORM OF THE INVERSE

1. Advantages

Due to the limited random access memory (RAM) available on the eight-bit microcomputer, tableau (matrix) simplex methods limit problem size. In order to solve "large" problems involving several hundred variables and constraints, we must store some of the data "off-line". This is done by reading and writing data to non-volatile memory diskettes.

COMPLETE TABLEAU REPRESENTATION

$$Y = \begin{array}{c} \begin{matrix} X_B & & X_N & & \end{matrix} \\ \left[\begin{array}{c|c|c} I & B^{-1}N & B^{-1}b \\ \hline 0 & c_N - c_B B^{-1} N & c_B B^{-1} b \end{array}\right] \\ \begin{matrix} \text{basic} \\ \text{variables} \end{matrix} \quad \begin{matrix} \text{non-basic} \\ \text{variables} \end{matrix} \quad \begin{matrix} \text{right hand} \\ \text{side} \end{matrix} \end{array} = \{y_{ij} | y_{i0}\}$$

Figure 1.

The simplex method using multipliers (DANTZIG) or the revised simplex method with product-form inverse affords the following computational advantages while providing the necessary intermediate data that can be efficiently read from and written to diskette data files (see [Ref. 1, p. 210]).

a. Less data is recorded from one iteration to the next, which permits more significant figures to be carried or a larger problem to be solved within the limited memory of the microcomputer.

b. Where the original data has a high percentage of zero coefficients, there are fewer multiplications. In the standard simplex method, each iteration requires the recording of at least (m+1)(n+1) entries. Here, however, by use of cumulative multiplications, the amount of recorded information is reduced to 2m+1 entries.

c. High speed core (RAM) storage requirements are reduced.

2. Elementary Matrices

Consider the tableau represented in Figure 1, and suppose Y is transformed by a pivot operation where the pivot column is:

$$Y_q = (y_{1q}, y_{2q}, \ldots, y_{mq})^T, \text{ with pivot element } y_{pq} .$$

The result of the transformation is the matrix, EY, where E is the elementary matrix:

$$E = \begin{bmatrix} 1 & 1 & 0 & \cdots & 0 & v_1 & 0 & \cdots & 0 \\ & 0 & 1 & \cdots & 0 & v_2 & 0 & \cdots & 0 \\ & & 0 & & & & \cdot & & \cdot \\ & \cdot & & & 1 & & \cdot & & \cdot \\ & \cdot & & & & v_p & \cdot & & \cdot \\ & \cdot & & & & & 1 & & 0 \\ 0 & 0 & & & 0 & v_m & 0 & \cdots & 1 \end{bmatrix}$$

where $v_i = -y_{iq}/y_{pq}$, $i \neq p$ and $v_p = 1/y_{pq}$.

Now, since the elementary matrix, E, is determined entirely by the elements of the pivot column, the remainder comprising the identity matrix, all that must be stored is the pivot row index number and the associated column vector. At any intermediate iteration, k, the product of these elementary matrices represents the inverse of the basis:

$$B_k^{-1} = E_k E_{k-1} E_{k-2} \cdots E_2 E_1 ,$$

where E_k is the elementary matrix corresponding to the k-th pivot operation.

Now we will augment the algorithm previously stated.

1. Since the basic solution $X_B = B^{-1}b = (E_k(E_{k-1}(\ldots E_1 b)))$, we can maintain X_B current at each iteration by simply multiplying (on the left) by the new elementary matrix for that iteration. This information is used in Step 2 and is represented by Y_0.

2. Calculating current relative costs, $r = c_N - c_B B^{-1} N$ can be represented by

$$r = c_N - \lambda N ,$$

where

$$\lambda = c_B B^{-1} \quad \text{("simplex multipliers")}$$

$$= (((c_B E_k) E_{k-1}) \cdots E_1) .$$

3. Once the pivot column is chosen, the current values of the elements of that column are required for Step 2. Calculate:

$$Y_q = (E_k (E_{k-1} \cdots (E_1 A_q)) .$$

E. DEGENERACY AND CYCLING

Degeneracy is encountered when one or more of the elements of the current solution (right hand side), $B^{-1}b$, become zero. Thus it is possible that more than one basis has the same coordinates, X. When degenerate solutions occur, we can no longer argue that the simplex procedure will necessarily terminate in a finite number of iterations, as is true in the non-degenerate case [Ref. 1, p. 100], because the value of the objective function will change by an amount equal to zero and it is conceivable that the same set of basic variables may recur. If we were to continue, with the same selection of x_q and x_{j_p} for each iteration as before, the same set of basic variables could recur after

k iterations, and again after $2k$ iterations, etc., indefintely. This recurrence of the same basis is called "cycling", [Ref. 6, pp. 68-69]. We choose to proceed with an algorithm that does not allow cycling.

Pivot selection rules exist which ensure completion of the simplex method within a finite number of iterations. The rule referred to by Bland as "a simple finite pivoting rule" [Ref. 3] is stated as follows:

1. Among all candidates to enter the basis, select the variable x_q having the lowest index, i.e., pivot on the column q determined by:

$$y_{0q} = \min\{y_{0j}: \ y_{0j} < 0\} .$$

2. Among all candidates to leave the basis, select the variable x_{j_p} having the lowest index, i.e., pivot in the row p determined by:

$$p = \min\{s: \ y_{sq} > 0 \text{ and } \frac{y_{s0}}{y_{sq}} = \min\{\frac{y_{i0}}{y_{iq}}: \ y_{iq} > 0\}\} .$$

Since we have added a second possible pivot option in the bounded variable simplex method, we must modify this last statement as follows:

$$p = \min\{s: \ \frac{y_{s0}}{y_{sq}} = \min\{\frac{y_{i0}}{y_{iq}}: \ y_{iq} > 0, \ \frac{y_{i0}-UB_{j_i}}{y_{iq}}: \ y_{iq} < 0, \ UB_{j_i} \text{ finite}\}\}$$

Note that in either case the row, p, contains the first occurrence of the minimum positive ratio.

It has been shown on small test problems that implementation of Bland's rule may cause a significant increase in the number of iterations required to complete the problem [Ref. 7]. Current research by Brown and Dewald [Ref. 8], suggests a hybrid rule that restricts the pricing rule only when the current solution is degenerate. Looking again at Step 1 of this rule we will expand the procedure as follows:

a. Define a permutation set of the column indices: $K = \{k_1, \ldots, k_j, \ldots, k_n\}$, with a partition after k_s.

b. Assign the partition boundary $s = 0$.

c. If the minimum positive ratio encountered in Step 2 of the previous pivot is non-zero, indicating that the current solution is non-degenerate, set $s = 0$ and select the variable x_q by the original most negative rule.

d. If the minimum positive ratio encountered in the previous step equals zero then the current solution is degenerate. Select x_q by Bland's rule #1 such that:

* 1. If $s = 0$ then $q = k_j$ where j is minimized:

$$\text{MIN}\{j: \; y_{0k_j} < 0, \; j = 1, \ldots, n\} .$$

* 2. If $s \neq 0$ then select x_q such that $q = k_j$ where j is minimized:

$$\text{MIN}\{j: \; y_{0k_j} < 0, \; j = 1,\ldots,s\} ,$$

or, if no such q exists, then select q such that

$$y_{0q} = \text{MIN}\{y_{0k_j}: \; y_{0k_j} < 0, \; j = s+1,\ldots,n\} .$$

e. Once the incoming variable has been chosen then if the "winning" index, $j > s$ interchange k_j and k_{s+1}, and assign $s \leftarrow s+1$.

What we have constructed is an ordering of the columns of the tableau such that Bland's rule is followed, but its pricing restriction is applied only when absolutely necessary. In this way "most negative pricing" is allowed whenever possible. Now we must reconstruct Bland's proof that the simplex method under this rule cannot cycle, hence is finite.

PROOF:

1. Since the simplex method cannot cycle as long as the minimum positive ratio > 0, then monotonicity of the objective function value implies that the simplex method terminates after finitely many pivots examine a finite number of ordered bases.

2. If the minimum positive ratio = 0, Bland's rule is used until:

   * i. the minimum positive ratio > 0.
   * ii. optimality is verified.
   * iii. primal unboundedness is discovered.

By Bland's rule this will occur in a finite number of pivots.

3. Once the pivot is completed for some ratio greater than zero, the simplex method can not revisit any previous basis. The algorithm has moved to a new basis corresponding to an improved value of the objective function.
4. Therefore, the monotonicity of the objective function value implies that the algorithm terminates in finitely many pivots.

F. REINVERSION

A characteristic of the product-form inverse algorithm is that with each pivot an increasing amount of work must be done in order to apply the elementary transformations. The addition of each elementary transformation vector increases the number of multiplications in the next iteration by as much as twice the number of constraints in the problem. At some point it becomes more efficient to replace the list of vectors, commonly referred to as (ETA), with a smaller set representing the same basis.

It is convenient to again transform the right hand side b' at this point to accommodate reflections of variables with upper bounds as follows:

- For each reflected variable x_j:

$$b''_i = b'_i - \sum_j a_{ij} * MUB_j, \quad \text{for all } i,$$

where a_{ij} are the original non-zero problem elements.

Hellerman and Rarrick [Ref. 2], present a statement of the reinversion problem as follows:

- Given--a set of basic variables
- Find--a set of transformation vectors (ETA) which imply the inverse of the basis in such as way as to:
  * a. minimize the number of non-zero elements in ETA and
  * b. minimize the work done in forming the ETA.

This is, of course, extremely expensive to do optimally.

Starting with Markowitz's observations [Ref. 9] on minimizing the number of non-zero elements when forming the ETA vectors, Hellerman and Rarrick develop a fast and efficient heuristic algorithm called the "preassigned pivot procedure". This development shows that if the rows and columns of the basis matrix can be re-ordered so that a pivot sequence can be assigned progressing down the diagonal of the transformed matrix M, where M is lower triangular with non-zero diagonal elements, then no additional non-zero elements are generated in the ETA representation.

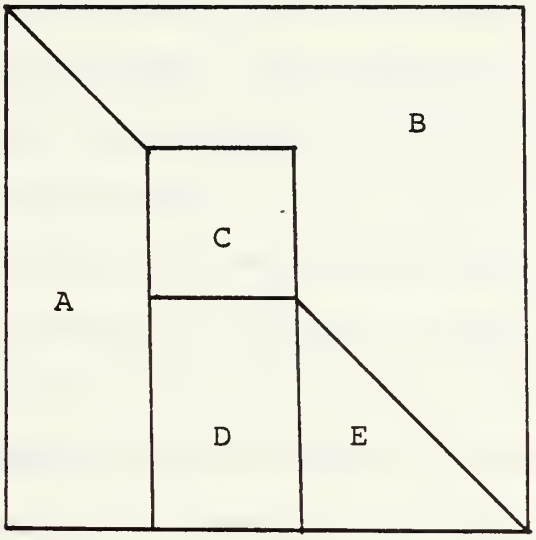

Figure 2.

In general, the lower triangular form cannot be achieved but only approximated. Some of the columns of M will have non-zero elements above the diagonal. These columns are called "spike columns". In fact at some point in the process there will usually remain a set of spike columns called the "bump" so that the matrix can be represented as shown in Figure 2.

In Sections A and E we have found pivots on the main diagonal and all other non-zero elements are below the diagonal. Section B has all elements equal to zero. Section C is the bump.

Note that Sections A and E have zero multipliers as we proceed down the diagonal. The major problem, then, is the build-up of non-zero elements in Sections C and D. This

build-up can be minimized by breaking Sections C and D into two or more bumps so that the non-zero build-up occurs only in the smaller bumps. The process of selection of each spike in the preassignment procedure is to choose the next pivot column so that:

a. when its effect is removed from the row counts (number of non-zero elements in each row not already assigned to the pivot sequence), it will create a maximum number of row counts of unity or, at least, as many small row counts become smaller as is possible, and

b. the chosen column can be pivoted as soon as possible (thereby being updated by the smallest subset of ETA).

The concept of a tally function is used in the algorithm to effect the above goals. The function is defined by:

- $t_k(n)$ = the number of non-zeros that column n has in rows whose row count is less than or equal to k, for all n {all columns not already assigned a position in the column pivot sequence or designated as a spike column}. The (k,n) combination giving the maximum $t_k(n)$ selects the pivot column.

Two other considerations are also mentioned in connection with the preassigned pivot procedure [Ref. 2]. If during the scan of column counts, a column with a count of zero is found, then a singularity exists in the current basis. The column should be dropped from the basis. Similarly, when

scanning the row counts, if a row count of zero is found in an unassigned row, then the associated constraint is redundant and could be removed from the problem or represented by a basic artificial with value zero. The other consideration is that in computer implementations, if the updated pivot element becomes too small (machine zero) to be used for a pivot then it is necessary to find an alternate pivot element. This can be done, in theory, by a proper choice of an alternate spike pivot column [Ref. 2] (a "spike swap"). Another method is to use Gaussian partial pivoting, find another row in the current column having a favorable pivot element and continue. This may lead to a compromise of our original goals and introduce additional spikes. A third alternative is to replace the current basic column with a logical variable (unit vector) column having unity in the pivot row. Note that this technique may introduce an infeasibility in which case a post-reinversion return to Phase I will be required. The initial implementation of this system will include all three options. Although the "spike swap" technique seems to be the preferred procedure it requires updating of multiple columns. For this reason the partial pivot procedure is tried first followed by the spike swap if no non-zero elements are found. When both of these techniques fail, the unit vector insertion is used as a last resort.

IV. IMPLEMENTATION

The implementation uses a KAYPRO II, 8-bit microcomputer with 64K random access memory and two single-sided, double density 5-1/4 inch floppy diskette drives. Approximately 57K of this memory is available for program loading and storage of variables. The diskette drives are used for subroutine storage as well as off-line storage of the problem files and the product-form representation of B^{-1}. The remaining 7K of internal memory is utilized by the operating system. The language in which the code is written is a "semi-standard" version of Pascal. The programming package, JRT PASCAL version 3.0, is very nearly a complete version of Pascal as initially designed by Wirth (e.g., [Ref. 10]). The JRT version has numerous extensions that make file handling on the microcomputer relatively simple. The major disadvantage of this language package [Ref. 11] is that the code is never completely compiled and, therefore, requires a resident "exec" driver which interprets the semi-compiled code. This exec occupies 24K of the usable memory and executes less efficiently than completely compiled object code. An additional disadvantage of this language-machine combination is that the KAYPRO II, without an available modification, has a Z-80 processor that runs at a speed of only 2.5 MHz. This relatively slow processor speed has an

obvious effect on solution times, and motivates the developer to organize programs in small, easily compiled externally linked Pascal procedures.

A. DATA STRUCTURES

The data structures used in this implementation are quite simple. They consist of a number of one-dimensional vectors of dynamic length, and data files which are recorded on non-volatile memory diskettes. The size of the vectors is determined by the number of variables and constraints of the problem. The size of the data files is dependent on the number of non-zero problem elements and the number of iterations (pivots) performed. To illustrate the structure a listing of arrays and variables follows:

- Major data types: Listed in Pascal format for convenience, the data types defined as records consist of two-dimensional arrays of elements that may be accessed with a single "read" statement. This Pascal convention is of great value when reading from and writing to off-line files.
  - * real
  - * integer
  - * boolean
  - * matrix = record

```
        a:  real;       non-zero problem element
        iar:  integer;  row index of the problem
                        element
```

* etavec = record

  etas: real; non-zero elementary matrix element

  ieta: integer; row index of the elementary matrix element.

* ranges = record

  lb, mub: real; lower bound for variable modified upper bound (UB - LB) for variable).

- Array variables:
  * bounds: array of type = ranges; variable bounds.
  * c: array of reals; initial cost coefficients.
  * e: array of integers; status of variables, basis or non-basic, or removed from consideration in the problem. Reflections are indicated by the sign of the element, negative indicating a reflected variable.
  * jbasic: array of integers; variable basic for the each constraint.
  * ka: array of integers; random record number of first element of each column. The random record number is the location key into a random access diskette file and indicates the logical record number at which to enter.
  * ke: array of integers; random record number of first eta element in the eta vector for each pivot.

* kj: array of integers; row number for each pivot.
* xb: array of reals; current right hand side, $B^{-1}b$.
* tc: array of reals; current column of the tableau, $B^{-1}A_q$.
* tl: array of reals; current simplex multipliers, $C_B B^{-1}$.
* rside: array of reals; initial [untransformed] right hand side, b.
* unitvec: array of integers; index of original logical column for row i.
* cycle: array of integers; hybrid Bland's rule vector for pricing with degeneracy.

- File variables: (dimensioned 2 by the number of file elements)
  * eta: type etavec
    a B inverse matrix element
    from etavec data type.
  * ele: type matrix
    a problem matrix element
    from matrix data type.

B. INPUT

Input to the problem solver is accomplished through the use of three subroutines. These include an interactive session, "TYPEPROB", during which prompts are given regarding

required data input procedures and options; a module, "MODPROB", which transforms the formulation into canonical form and sets certain vector parameters, and a "READPROB" subroutine which creates the final formatting of the problem.

Initial problem input may also be created using any simple word processor and the first subroutine, "TYPEPROB", may be omitted by menu selection. The format required for the file is illustrated in Appendix A.

The problem data used in the example throughout this chapter is taken from the bounded variable example of Luenberger [Ref. 4], page 52. This problem statement is included in Appendix A.

1. TYPEPROB

This subroutine is designed to create a diskette text file of the problem formulation. An interactive, menu driven series of prompts is used to explain the input requirements and options of the input system. Appendix B displays a sample input session.

Upon answering the first question posed by the program with (1), to input a new problem, the user will be asked to specify a problem name. This name must be EXACTLY eight characters long. In the current implementation this is a file-naming restriction.

The prompt will then present a series of text pages. Appendix B illustrates these pages. When the user completes his responses to these questions, a series of requests will be presented as follows:

INPUT NUMBER OF CONSTRAINT ROWS
Do not count the objective function.

The user inputs the integer representation of the number of constraint rows of the problem formulation.

INPUT NUMBER OF VARIABLES
Do not count logical variables.
DO NOT count the right hand side as a column.

At this point more instructions will be given on the proper procedure for input of integer and real data types. Then, the columns will be accepted from the user, one column at a time. The variable associated with the column will first be named. This name may contain up to 5 alpha-numeric characters. The next question posed will be a multiple-choice menu of variable bounds for the current variable. The choices are:

(1) 0 $\leq$ VAR < infinity

(2) -infinity < VAR $\leq$ 0

(3) VARIABLE IS UNRESTRICTED (free)

(4) a $\leq$ VAR $\leq$ b

(5) a $\leq$ VAR < infinity

(6) -infinity < VAR $\leq$ b.

Simply choose the appropriate category for the current variable. If the variable is bounded, then the

next entries will be the lower bound entry, and/or upper bound entry as appropriate.

At this time the following "heading" will be presented on the screen.

```
ROW#  /  VALUE   //   OPTIONAL ROW #  /   VALUE
Negative row to end column.
```

The user is to input the INTEGER row number followed by the REAL value of the NON-ZERO element in the specified column and row. An additional row number and value is allowed as long as the column number does not change. ALL ROWS OF THE CURRENT COLUMN MUST BE INPUT AT THIS POINT. This includes the objective coefficient for this column as well as the coefficients of the column's constraint elements. Additionally, more than one objective coefficient may be entered at this time. This is to allow for maximum and minimum problems to be entered using the same constraints but different objective functions.

When a -1 is entered in the next row entry position, the column will be terminated and the next column will be presented. When all variable columns have been entered the format of the entries will be changed to outline the input required for the right hand side entries. This format will be:

```
For RHS #1, ROW # 1
          ENTER  G   L   or E
          FOR   >=  <=      =
FOLLOWED BY:
     <SPACE>,   value of RHS,   <RETURN>..
```

This prompt will be presented for each row. The next prompt will then ask if another RHS column is to be entered. Thus, multiple right hand side columns are accommodated. In this way multiple problems using the same matrix, A, do not need to be re-entered for each objective and right hand side that might apply. When no more right hand side columns are required the subroutine will terminate and the program will request information concerning the objective function and right hand side to be considered for the imminent solution.

The file that results from this subroutine is stored on the diskette in the form shown in Appendix A. The name of the file is B:probname.TMP, where "probname" is the eight character name entered by the user. If this file is constructed manually without the use of the subroutine, then the appropriate name must be given to the file so that the program can find it on the "B:" diskette in the future. This file-naming procedure which includes the diskette index is not specific to the KAYPRO II but is endemic to microcomputers.

2. MODPROB

At the termination of the subroutine "TYPEPROB" or if the selection is made to re-run a problem that has been previously entered, then the subroutine "MODPROB" will be called. This subroutine will modify the format of the problem file to include only that right hand side and that

objective function applicable to this specific problem. It will also add the appropriate logical variables to the column list. This new set of non-zero problem elements will then be written to a new text file, B:probname.DAT, for use. Additionally, the first set of basic variables will be listed, by column number, with negative column numbers representing artificial variables and indicating that Phase I simplex will be required. An example of this DATa file is shown in Appendix C.

This subroutine also allocates dynamic storage for vectors and writes a file listing variable names. The user will notice a delay during the time "MODPROB" is working. A large portion of that time is due to the naming of logical variables. The naming routine is slow in this implementation due to inefficiencies in the JRT PASCAL structure (not an important consideration in the development of the basic algorithms). There are faster ways to name the logical variables, but a better method was not found for JRT PASCAL.

3. READPROB

This subroutine completes the reading of the problem data file into a working file that is random access, binary and unreadable to a text editor. This is a fast access off-line file from which the appropriate non-zero elements of a column can be accessed when a column update is required during the simplex procedure. Upon termination of the "READPROB" subroutine, the problem has been transformed

to canonical form and all initial values have been set. The problem is now ready for the simplex technique.

C. THE PROBLEM SOLVER

The simplex algorithm implemented is a "textbook" Pascal translation of the theory and approach already discussed. A modular, procedure-calling technique is used which allows compilation of small units of code, linked as external Pascal procedures. A short description of each procedure follows.

1. SIMPLEX

The simplex procedure is the driver for the problem solver. It determines the requirement for Phase I or Phase II, initializes the required objective (cost) vector and calls all of the other procedures directly associated with the simplex algorithm.

2. PHASEI

This procedure solves the modified problem

$$\text{Minimize} \quad \sum_{j \text{ artificial}} x_j$$

subject to the given constraints,

arriving at a first feasible solution. If no such solution exists and the problem is infeasible then the most feasible, last iteration solution is output and the program is terminated. If a feasible solution is found, then the original problem cost vector is restored and Phase II simplex is begun.

3. BTRAN

The "BTRAN" subroutine calculates the simplex multipliers

$$\lambda = c_B B^{-1}$$

using the formula $(\ldots(((c_B E_k) E_{k-1})\ldots)E_1)$.

A copy of the Pascal code for "BTRAN" is included in Appendix D as an example of the implementation code.

4. CHUZQ

After "BTRAN" computes the simplex multipliers, this procedure is called to calculate the current reduced costs for all non-basic variables, $c_N - \lambda N$. In the absence of degeneracy the most negative reduced cost over all X_N is chosen, resulting in the most rapid convergence to the optimal solution. In the presence of degeneracy the hybrid implementation of Bland's rule number 1 is activated.

5. FTRAN

This procedure is called at any time that a column vector update is required. The function calculates, $Y_q = B^{-1}A_q$ using the formula

$$E_k(E_{k-1}(\ldots (E_1 A_q))\ldots) .$$

6. CHUZP

Using an updated column from "FTRAN", "CHUZP" determines the pivot row, p, using the three-case test for bounded variables.

7. PIVOT

This procedure is called by the simplex subroutine and by the "REINVERT" procedure. Each time "PIVOT" is called, an asterisk (*) is displayed on the CRT for reference. Each asterisk signifies the formation of one ETA vector in the B^{-1} product-form. If case a of the bounded variable simplex algorithm is encountered and a non-basic variable is reflected, (no eta vector is generated), then a pound sign (#) is printed in place of the asterisk.

8. REINVERT

"REINVERT" is a direct implementation of Hellerman and Rarick's preassigned pivot procedure with a few modifications. Forward pivots are completed as they are assigned so that subsequent forward transformations can be used immediately to reveal scaling difficulties requiring spike swapping (pivot element too small). Constraint redundancy checks are also implemented as described in [Ref. 2], page 214.

D. OUTPUT

The "FILEOUT" procedure causes the current solution to be written to the output file, B:PFI.LST. At program termination this text file may be printed using any simple word processor. An example of the program output for the example problem is shown in Appendix E.

This procedure reverses most transformations used to convert to (LPC). Thus, upper and lower variable bounds,

free variables, and the extremal operator (min/max) appear on the report as they did in the original formulation. Constraints appear with non-negative right hand sides. The report also lists reduced costs for non-basic variables and dual prices for constraints with structural variables basic.

V. CONCLUSIONS AND RECOMMENDATIONS

The simplex implementation described has shown that advanced algorithms for linear programming problems can be packaged in an easy to use, interactive system on a microcomputer. It has also shown that while solution time is certainly not on the same order of magnitude offered by mainframes, neither is there the cost associated with mainframe CPU time-sharing. Reasonable solution time for a linear program on a microcomputer might be the length of a coffee break. This implementation averages three to four seconds per pivot for early iteration pivots and approximately 5 seconds after thirty pivots.

A time test was run on a problem posed by MICRO VISION (135 Herzel Blvd., Lindenhurst, N.Y. 11757) as an advertisement for their "MATHEMATICAL PROGRAMMING PACKAGE II". This problem restated as a bounded variable problem has 8 constraints and 17 variables, including logicals. The original A-matrix is 85 percent dense. Total solution time on the KAYPRO-II at 2.5 MHz clock speed was 125 seconds. This time included 30 seconds required to write three solutions, Phase I, Phase II, and the reinversion solution. Phase I and reinversion times were also included in the solution time. While these times are not as good as the MICRO VISION time published, the limit on the number of variables and constraints

for the MICRO VISION version is listed as "100 x 100 on the IBM Personal Computer or model XT with 128K memory". The limit on constraints and variables for the description given in this paper has not been reached. The limiting factors are the number of elements in the one-dimensional vectors required for the underlying data structure and the size and density of the diskettes used to store the out-of-core files.

Serial file organization of the problem and eta files on diskettes would *greatly* improve performance. This modification would require organization of these files in "pages" of columns to permit efficient serial reading of a (problem dependent) set of columns at each diskette access. In concert with this modification, partial pricing ("batch pricing") would probably improve execution efficiency a bit more. Unfortunately, these relatively easy modifications require significant redesign of dynamic memory management and file handling constraints. These enhancements have not been implemented at this writing.

We hope that the work presented here will further stimulate the development of additional mathematical programming software for use on microcomputers. As costs of microcomputers continue to decrease while system capabilities progress, the operations research community must be prepared to take full advantage of the availability and potential of these valuable tools.

Copies of the PASCAL code and diskettes formatted for the KAYPRO II, containing all subroutines may be obtained from the author. Please address requests to:

Major D. W. Theune
P. O. Box 1083
Springfield, Va., USA 22151.

APPENDIX A

EXAMPLE INPUT FILE

PROBLEM STATEMENT

$$\text{minimize } 2x_1 + x_2 + 3x_3 - 2x_4 + 10x_5$$

$$\text{subject to } x_1 + x_3 - x_4 + 2x_5 = 5$$

$$x_2 + 2x_3 + 2x_4 + x_5 = 9$$

$$0 \leq x_1 \leq 7,\ 0 \leq x_2 \leq 10,\ 0 \leq x_3 \leq 1,\ 2 \leq x_4 \leq 5,\ 0 \leq x_5 \leq 3$$

EXAMPLE INPUT FILE

```
2  5                number of constraints / number of variables
2  1  1   3  2          < no blank space here >
2  2  1   3  1
3  1  1   2  2
3  3  3               non-zero problem elements
4  1  -1  2  2
4  3  -2                    format:
5  1  2   2  1                              /   optional
5  3  10               col # / row # / value / row # / value
                      < blank space required >
 1 e 5                right hand sides
 1 e 9                      rhs # / restriction / value
                      < blank space required >
X1      4
X2      4
X3      4               variable name / type of bounds
X4      4
X5      4
```

```
                < blank space required >
UB X1     7
UB X2     10
UB X3     1        variable bounds UB = upper or LB = lower
LB X4     2             variable name and value
UB X4     5
UB X5     3
```

APPENDIX B

INTERACTIVE SESSION

CRT SCREEN PRESENTATION 1:

```
A>b:exec pfi
Exec  ver 3.Ø

DEBUG?   Ø = no,    1 = yes
Ø
Do you wish to:
      (1) input a new problem?
      (2) re-run an old problem with modifications?

                TYPE YOUR CHOICE   1 or 2
1
Input the problem name.
This name will be used whenever the problem is recalled
FORMAT:
        enter EXACTLY eight characters.   CCCCCCCC
bounded2
```

SCREEN 2

```
        THIS PROGRAM IS INTENDED TO BUILD A DATA FILE FOR PRESENTATION OF
                    A LINEAR PROGRAM TO THE PACKAGE THESIS.PFI


If you want to continue type "go" and <enter>.
If you have already entered your data
   or if the data file you wish to use has already been created
   then type "return" and <enter> to return to the main program.

go
```

SCREEN 3

```
        THIS PROGRAM IS INTERACTIVE:
                PROMPTS WILL BE GIVEN AS FOLLOWS:

        The first two entries will be the NUMBER OF ROWS and COLUMNS.

        Remaining entries will be entered in a modified column input format.
    Columns will be requested in order.
    You will enter the current row number of the next non-zero value
    and the value associated with that column and row.

          Row #  /  Value  //  Optional Row #  /  Value

NOTE::
    Current column numbers will be provided. If all rows for the current
    column have been entered, type "-1" for the next row number and the
    column number will be incremented.


TYPE ANY CHARACTER and <enter> TO CONTINUE
f
```

SCREEN 4

```
        INPUT NUMBER OF CONSTRAINT ROWS
                    Do not count the objective function!!!!
        2



        INPUT NUMBER OF VARIABLES
                DO NOT COUNT LOGICAL VARIABLES!!!!
         and    DO NOT ENTER LOGICAL VARIABLE COLUMNS!!!!!!

           DO NOT COUNT THE RIGHT HAND SIDE AS A COLUMN!!!!


        5
```

SCREEN 5

```
YOU WILL NOW BE ASKED TO ENTER THE NON-ZERO PROBLEM ELEMENTS.
All entries will be entered by column.
A RESTRICTED number of rows and columns may be entered.

The first  2 rows of each column represent the non-objective
rows. All additional rows entered will be treated as additional
objective rows. You will be asked later, which objective function
is to be considered in a given problem solution.
A MAXIMUM OF 10 OBJECTIVE ROWS MAY BE ASSIGNED TO A GIVEN PROBLEM.

SIMILARLY:
The first  5 columns will be treated as variables. All remaining
columns will be assumed as independent sets of technological constraints.
You will be asked later, which of these sets is to be considered for
the current problem solution.

TYPE ANY CHARACTER and <enter> TO CONTINUE

m
```

SCREEN 6 (DATA INPUT)

```
The following entries may pe placed IN ORDER in any column.
The following restrictions apply:
   All row numbers must be entered as integers.
   All values must be entered as real numbers as follows;
         0.123 or 2.34 or 34.0 . The decimal must have a preceeding
                                 and a following numeral.

ENTER NOW:

Current column is 1
Type up to 5 characters to assign variable name for column  1
X1

Choose appropriate variable bounds for variable X1

                 LB                          UB
(1)      0       <=  X1        <=    infinity
(2)      -infinity    <=  X1        <=     0
(3)      X1    unrestricted (FREE)
(4)         a        <=    X1        <=       b
(5)         a        <=    X1        <=    infinity
(6)      -infinity   <=    X1        <=       b

TYPE (1 or 2 or ... 6)
4
     WHAT IS THE LOWER BOUND?  a = LB.
0
     WHAT IS THE UPPER BOUND?   b = UB.
7
```

```
   ROW #  /  VALUE  //  OPTIONAL ROW #  /  VALUE
   Negative row to end column
1 1 3 2
Last row entered was objective row # 1
-1
Current column is 2
Type up to 5 characters to assign variable name for column  2
X2

Choose appropriate variable bounds for variable X2

                LB                          UB
(1)     Ø       <=  X2      <=   infinity
(2)     -infinity   <=  X2      <=    Ø
(3)     X2   unrestricted (FREE)
(4)        a       <=   X2      <=      b
(5)        a       <=   X2      <=   infinity
(6)     -infinity <=   X2      <=      b

TYPE (1 or 2 or ... 6)
4
     WHAT IS THE LOWER BOUND?  a = LB.
Ø
     WHAT IS THE UPPER BOUND?   b = UB.
1Ø
   ROW #  /  VALUE  //  OPTIONAL ROW #  /  VALUE
   Negative row to end column
2 1 3 1
Last row entered was objective row # 1
-1
Current column is 3
Type up to 5 characters to assign variable name for column  3
X3



Choose appropriate variable bounds for variable X3

                LB                          UB
(1)     Ø       <=  X3      <=   infinity
(2)     -infinity   <=  X3      <=    Ø
(3)     X3   unrestricted (FREE)
(4)        a       <=   X3      <=      b
(5)        a       <=   X3      <=   infinity
(6)     -infinity <=   X3      <=      b

TYPE (1 or 2 or ... 6)
4
     WHAT IS THE LOWER BOUND?  a = LB.
Ø
     WHAT IS THE UPPER BOUND?   b = UB.
1
   ROW #  /  VALUE  //  OPTIONAL ROW #  /  VALUE
   Negative row to end column
1 1 2 2
3 3 -1
Current column is 4
Type up to 5 characters to assign variable name for column  4
X4
```

```
Choose appropriate variable bounds for variable X4

                LB                          UB
(1)      Ø       <=  X4       <=   infinity
(2)      -infinity   <=  X4       <=    Ø
(3)      X4   unrestricted (FREE)
(4)         a       <=   X4       <=       b
(5)         a       <=   X4       <=   infinity
(6)      -infinity <=   X4       <=       b

TYPE (1 or 2 or ... 6)
4
     WHAT IS THE LOWER BOUND?  a = LB.
2
     WHAT IS THE UPPER BOUND?   b = UB.
5
   ROW #  /  VALUE  //  OPTIONAL ROW #  /  VALUE
   Negative row to end column
1 -1 2 2
3 -2 -1
Current column is 5
Type up to 5 characters to assign variable name for column  5
X5

Choose appropriate variable bounds for variable X5

                LB                          UB
(1)      Ø       <=  X5       <=   infinity
(2)      -infinity   <=  X5       <=    Ø
(3)      X5   unrestricted (FREE)
(4)         a       <=   X5       <=       b
(5)         a       <=   X5       <=   infinity
(6)      -infinity <=   X5       <=       b

TYPE (1 or 2 or ... 6)

4
     WHAT IS THE LOWER BOUND?  a = LB.
Ø
     WHAT IS THE UPPER BOUND?   b = UB.
3
   ROW #  /  VALUE  //  OPTIONAL ROW #  /  VALUE
   Negative row to end column
1 2 2 1
3 1Ø -1
Current column is RIGHT HAND SIDE # 1


All rows of RIGHT HAND SIDE require an entry.
ENTER type of constraint followed by the value of the current RHS.
ZERO VALUES AS Ø.Ø

For RHS # 1, ROW # 1
                ENTER   G    L   or  E
                FOR     >=   <=       =
Followed by:
     <SPACE> ,  value of RHS ,  <RETURN>..
e 5
```

```
For RHS # 1, ROW # 2
                ENTER   G    L   or  E
                FOR    >=   <=       =
Followed by:
     <SPACE> ,  value of RHS ,  <RETURN>..
e 9

RHS 1 is complete. Do you have another RHS column?    ( Y or N )
( Y or N )
n
Is the objective to be MINimized        : or MAXimized.
ENTER        MIN    or   MAX
min

Input the integer number of the objective row to be considered.
    This integer must be in the range   1 to 10
1

Input the integer number of the Right Hand Side to be considered.
    This integer must be in the range 1 to number of RHS columns.
1
Begin phase  1
***dealing with degeneracy
*dealing with degeneracy

Phase I solution complete.

Do you wish to print out all intermediate solutions?
Type  any positive integer if YES
          negative integer or 0   if NO
0
Begin phase  2
dealing with degeneracy
*dealing with degeneracy
****
Phase II solution complete:

REINVERSION IN PROGRESS

**
REINVERSION COMPLETE


Program termination
```

APPENDIX C

EXAMPLE PROBLEM FILE

```
2  7                          m,n      Constraints, variables
1  1  1    3  2
2  2  1    3  1                       This list is produced by
3  1  1    2  2                       the package and is not
3  3  3                               accessible to the user.
4  1  -1   2  2
4  3  -2
5  1  2    2  1
5  3  10
6  1  1
7  2  1
8  1  5
8  2  9

-6                          initial basis columns (-) denotes artificials
-7
```

APPENDIX D

BTRAN PROCEDURE LISTING

```
procedure btran;
{produce pricing vector tl[i]= Cb * Binverse}

   var
      i, j, ki: integer;
      tx: real;

   begin
      for i:= 1 to m do
         begin
            tl[i]:= c[jbasic[i]];
            if (e[jbasic[i]]<0) then tl[i]:= -tl[i];
            if (abs(tl[i])<zl) then tl[i]:= 0.0;
         end;
      if (np <> 0) then
           begin
              for l:= np downto 1 do
                 begin
                      tx:= 0.0;
                      for ki:= ke[l] to ke[l+1]-1 do
                         begin
                              read(binv, rrn, ki; eta);
                              tx:= tx + tl[eta.ieta] * eta.etas;
                         end;
                      if (abs(tx) < zl) then tx:= 0.0;
                      tl[kj[l]]:= tx;
                 end;
           end;
      if (debug) then
         begin
            writeln(outfile; 'BASIC COST VECTOR');
            for j:= 1 to m do
               writeln(outfile; tl[j],'  ');
         end;
   end; {procedure btran}.
```

APPENDIX E

OUTPUT LISTING FOR LP PROBLEM SOLVER

```
   Output for problem bounded2

Dealing with degeneracy!

Dealing with degeneracy!

   Phase I solution :
   The problem is minimize.

   Variable      Basic        C[j]        Value                    Reduced
     Name          for      initial                                 Cost
                Constraint

   X1      :          1        0      0.0000000000              0.0000000000
   X2      :          2        0      0.0000000000              0.0000000000
   X3      :                   0      1.0000000000              0.0000000000
   X4      :                   0      2.0000000000              0.0000000000
   X5      :                   0      3.0000000000              0.0000000000


   Row      Basic   Original          Slack                     Dual
            Variable  RHS             Value                     Prices

    1       X1       = 5             0.0000000000              0.0000000000
    2       X2       = 9             0.0000000000              0.0000000000


   Value of the objective function :   0

   Current value and solution represent tableau for pivot # 2

Dealing with degeneracy!

Dealing with degeneracy!
```

Optimal phase II solution :
The problem is minimize.

Variable Name		Basic for Constraint	C[j] initial	Value	Reduced Cost
X1	:		2	7.0000000000	2.0000000000
X2	:	1	1	1.0000000000	0.0000000000
X3	:		3	1.0000000000	3.0000000000
X4	:	2	-2	3.0000000000	0.0000000000
X5	:		10	0.0000000000	1.0000000000

Row	Basic Variable	Original RHS	Slack Value	Dual Prices
1	X2	= 5	0.0000000000	-4.000000000
2	X4	= 9	0.0000000000	-1.000000000

Value of the objective function : 12

Current value and solution represent tableau for pivot # 5

REINVERSION AFTER PIVOT # 5

REINVERSION COMPLETE The problem is minimize.

Variable Name		Basic for Constraint	C[j] initial	Value	Reduced Cost
X1	:		2	7.0000000000	2.0000000000
X2	:	2	1	1.0000000000	0.0000000000
X3	:		3	1.0000000000	3.0000000000
X4	:	1	-2	3.0000000000	0.0000000000
X5	:		10	0.0000000000	1.0000000000

Row	Basic Variable	Original RHS	Slack Value	Dual Prices
1	X4	= 5	0.0000000000	-4.0000000000
2	X2	= 9	0.0000000000	-1.0000000000

Value of the objective function : 12

Current value and solution represent tableau for pivot # 2

LIST OF REFERENCES

1. Dantzig, G. B., Linear Programming and Extensions, Princeton University Press, 1963.

2. Hellerman, E. and Rarick, D., "Reinversion with the Preassigned Pivot Procedure," Mathematical Programming, V. 1, Pp. 195-216, 1971.

3. Bland, R. G., "New Finite Pivoting Rules for the Simplex Method," Mathematics of Operations Research, V. 2, No. 2, Pp. 103-107, May 1977.

4. Luenberger, D. G., Introduction to Linear and Nonlinear Programming, Addison-Wesley, 1973.

5. Orchard-Hays, W., Advanced Linear-Programming Computing Techniques, McGraw-Hill, 1968.

6. Murtagh, B. A., Advanced Linear Programming: Computation and Practice, McGraw-Hill, 1981.

7. Avis, D. and Chvatal, V., "Notes on Bland's Pivoting Rule," Mathematical Programming Study, V. 8, Pp. 24-34, 1978.

8. Dewald, L. (Private Communication, September 1983).

9. Markowitz, H. N., "The Elimination Form of the Inverse and Its Application to Linear Programming," Management Science, 3, No. 3, Pp. 255-269, 1957.

10. Jensen, K. and Wirth, N., Pascal User's Manual and Report, Springer-Verlag, 1976.

11. JRT Systems, JRT Pascal User's Guide, JRT Systems, 1983.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943	2
3.	Department Chairman, Code 55 Department of Operations Research Naval Postgraduate School Monterey, California 93943	1
4.	Professor Gerald G. Brown, Code 55Bw Department of Operations Research Naval Postgraduate School Monterey, California 93943	20
5.	Professor R. Kevin Wood, Code 55Wd Department of Operations Research Naval Postgraduate School Monterey, California 93943	1
6.	Professor Lawrence Bodin BMGT University of Maryland College Park, MD 20742	1
7.	MAJOR Donald W. Theune 10702 Fred's Oak Court Burke, VA 22015	5
8.	Headquarters United States Marine Corps Code RDRS-40 Washington, D.C. 20380	1

217769

Thesis
T3632 Theune
c.1 A microcomputer based general linear programming optimization package
26 pages
33436

217769

Thesis
T3632 Theune
c.1 A microcomputer based general linear programming optimization package.

thesT3632
A microcomputer based general linear pro
3 2768 001 01084 6
DUDLEY KNOX LIBRARY